Readlink

Vulnerable to TOCTOU issues

Sean Barnum, Cigital, Inc. [vita[1]]

2007-04-02

Part "Original Cigital Coding Rule in XML"

Mime-type: text/xml, size: 5623 bytes

<table>
<tr><td><b>Attack Category</b></td><td>• Path spoofing or confusion problem</td></tr>
<tr><td><b>Vulnerability Category</b></td><td>• Indeterminate File/Path
• TOCTOU - Time of Check, Time of Use</td></tr>
<tr><td><b>Software Context</b></td><td>• Filename Management</td></tr>
<tr><td><b>Location</b></td><td></td></tr>
<tr><td><b>Description</b></td><td><p>The readlink() function attempts to get the filename of the file pointed to by the given link. It is generally vulnerable to classic TOCTOU attacks.</p><p>A call to readlink() should be flagged if the first argument (the link) is used earlier in a check-category call.</p></td></tr>
<tr><td><b>APIs</b></td><td><table><tr><th>Function Name</th><th>Comments</th></tr><tr><td>readlink</td><td></td></tr></table></td></tr>
<tr><td><b>Method of Attack</b></td><td><p>The key issue with respect to TOCTOU vulnerabilities is that programs make assumptions about atomicity of actions. It is assumed that checking the state or identity of a targeted resource followed by an action on that resource is all one action. In reality, there is a period of time between the check and the use that allows either an attacker to intentionally or another interleaved process or thread to unintentionally change the state of the targeted resource and yield unexpected and undesired results.</p><p>The readlink() call is a use-category call, which when preceded by a check-category call can be indicative of a TOCTOU vulnerability.</p><p>A TOCTOU attack in regards to readlink() can occur when</p><p>a. A check for the existence of the file occurs or a non-fd reference (pathname) to the filename occurs</p><p>b. An actual call to readlink occurs.</p><p>Between a and b, an attacker could, for example, link the referenced file to a known file. The subsequent</p></td></tr>
</table>

[1] http://buildsecurityin.us-cert.gov/bsi-rules/35-BSI.html (Barnum, Sean)

<table>
<tr><td></td><td>readlink() call would have an unintended effect or impact.</td></tr>
<tr><td><b>Exception Criteria</b></td><td></td></tr>
<tr><td><b>Solutions</b></td><td>
<table>
<tr><th>Solution Applicability</th><th>Solution Description</th><th>Solution Efficacy</th></tr>
<tr><td>Generally applicable to any readlink.</td><td>Utilize a file descriptor version of stat/ fstat when checking.</td><td>Effective.</td></tr>
<tr><td>Generally applicable to any readlink.</td><td>The most basic advice for TOCTOU vulnerabilities is to not perform a check before the use. This does not resolve the underlying issue of the execution of a function on a resource whose state and identity cannot be assured, but it does help to limit the false sense of security given by the check.</td><td>Does not resolve the underlying vulnerability but limits the false sense of security given by the check.</td></tr>
<tr><td>Generally applicable to any readlink.</td><td>Limit the interleaving of operations on files from multiple processes.</td><td>Does not eliminate the underlying vulnerability but can help make it more difficult to exploit.</td></tr>
<tr><td>Generally applicable to any readlink.</td><td>Limit the spread of time (cycles) between the check and use of a resource.</td><td>Does not eliminate the underlying vulnerability but can help make it more difficult to exploit.</td></tr>
</table>
</td></tr>
</table>

Generally applicable to any readlink.	Recheck the resource after the use call to verify that the action was taken appropriately.	Checking the status after the operation does not change the fact that the operation may have been exploited but it does allow halting of the application in an error state to help limit further damage.

Signature Details	int readlink(const char *filename, char *buffer, size_t size);

Examples of Incorrect Code	`int check_status;`
	`int use_status;`
	`struct stat statbuf;`
	`...`
	`check_status=lstat(filename, &statbuf);`
	`...`
	`readlink(filename, buffer, buffer_size);`
	`...`

Examples of Corrected Code	`char *readlink_malloc (const char *filename)`
	`{`
	`int size = 100;`
	`while (1) {`
	`char *buffer = (char *) malloc (size);`
	`int nchars = readlink (filename, buffer, size);`
	`if (nchars < 0)`
	`return NULL;`
	`if (nchars < size) {`
	`buffer[nchars] = '\0';`
	`return buffer;`
	`}`
	`free (buffer);`
	`size *= 2;`
	`}`
	`}`

Source Reference	• ITS4 Source Code Vulnerability Scanning Tool [2]
Recommended Resource	

Discriminant Set	**Operating System**	• UNIX (All)
	Languages	• C • C++

Cigital, Inc. Copyright

The Build Security In (BSI) portal is sponsored by the U.S. Department of Homeland Security (DHS), National Cyber Security Division. The Software Engineering Institute (SEI) develops and operates BSI. DHS funding supports the publishing of all site content.

[1] mailto:copyright@cigital.com